

Efficient computation of exact solutions for quantitative model checking

Sergio Giro

Department of Computer Science, University of Oxford, Oxford, OX1 3QD, UK*

Quantitative model checkers for Markov Decision Processes typically use finite-precision arithmetic. If all the coefficients in the process are rational numbers, then the model checking results are rational, and so they can be computed exactly. However, exact techniques are generally too expensive or limited in scalability. In this paper we propose a method for obtaining exact results starting from an approximated solution in finite-precision arithmetic. The input of the method is a description of a scheduler, which can be obtained by a model checker using finite precision. Given a scheduler, we show how to obtain a corresponding basis in a linear-programming problem, in such a way that the basis is optimal whenever the scheduler attains the worst-case probability. This correspondence is already known for discounted MDPs, we show how to apply it in the undiscounted case provided that some preprocessing is done. Using the correspondence, the linear-programming problem can be solved in exact arithmetic starting from the basis obtained. As a consequence, the method finds the worst-case probability even if the scheduler provided by the model checker was not optimal. In our experiments, the calculation of exact solutions from a candidate scheduler is significantly faster than the calculation using the simplex method under exact arithmetic starting from a default basis.

1 Introduction

Model checking of Markov Decision Processes (MDPs) has been proven to be a useful tool to verify and evaluate systems with both probabilistic and non-deterministic choices. Given a model of the system under consideration and a qualitative property concerning probabilities, such as “the system fails to deliver a message with probability at most 0.05”, a model checker deduces whether the property holds or not for the model. As different resolutions of the non-deterministic choices lead to different probability values, verification techniques for MDPs rely on the concept of *schedulers* (also called policies, or adversaries), which are defined as functions choosing an option for each of the paths of an MDP. Model-checking algorithms for MDPs proceed by reducing the model-checking problem to that of finding the maximum (or minimum) probability to reach a set of states under all schedulers [4].

Different techniques for calculating these extremal probabilities exist: for an up-to-date tutorial, see [9]. Some of them (for instance, value iteration) are approximate in nature. If all the coefficients in the process are rational numbers, then the model checking results are rational, and so they can be computed exactly. However, exact techniques are generally too expensive or limited in scalability. Linear programming (LP) can be used to obtain exact solutions, but in order to achieve reasonable efficiency it is often carried out using finite-precision, and so the results are always approximations. (We performed some experiments showing how costly it is to compute exact probabilities using LP without our method.)

*This work was supported by DARPA and the Air Force Research Laboratory under contract FA8650-10-C-7077 (PRISMATIC). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of AFRL or the U.S. Government.

In addition, the native operators in programming languages like Java have finite precision: the extension to exact arithmetic involves significant reworking of the existing code.

We propose a method for computing exact solutions. Given any approximative algorithm being able to provide a description of a scheduler, our method shows how to extend the algorithm in order to get exact solutions. The method exploits the well-known correspondence between model-checking problems and linear programming problems [4], which allows to compute worst-case probabilities by computing optimal solutions for LP problems. We do not know of any similar approach to get exact values. This might be due to the fact that, from a purely theoretical point of view, the problem is not very interesting, as exact methods exist and they are theoretically efficient: the problem is that, in practice, exact arithmetic introduces significant overhead.

The simplex algorithm [5] for linear programming works by iterating over different *bases*, which are submatrices of the matrix associated to the LP problem. Each basis defines a *solution*, that is, a valuation on the variables of the problem. The simplex method stops when the basis yields a solution with certain properties, more precisely, a so-called *feasible* and *dual feasible* solution. By algebraic properties, such a solution is guaranteed to be optimal.

The core of our method is the interpretation of the scheduler as a basis for the linear programming problem. Given a scheduler complying with certain natural conditions, a basis corresponding to the scheduler can be used as a starting point for the simplex algorithm. We show that, if the scheduler is optimal, then the solution associated to the corresponding basis is feasible and dual feasible, and so a simplex solver provided with this basis needs only to check dual feasibility and compute the solution corresponding to the basis. As our experiments show, these computations can be done in exact arithmetic without a huge impact in the overall model-checking time. In fact, using the dual variant of the simplex method, the time to obtain the exact solution is less than the time spent by value iteration. If the scheduler is not optimal, the solver starts the iterations from the basis. This is useful for two reasons: we can let the simplex solver finish in order to get the exact solution; or, once we know that we are not getting the optimal solution, we can perform some tuning in the model checker as, for instance, reduce the convergence threshold (we also show a case in which the optimal scheduler cannot be found with thresholds within the 64-bit IEEE 754 floating point precision).

The correspondence between schedulers and bases is already known for discounted MDPs (see, for instance [7]). We show the correspondence for the undiscounted case in case some states of the system are eliminated in preprocessing steps. The preprocessing steps we consider are usual in model checking [9]: given a set of target states, one of the preprocessing algorithms removes the states that cannot reach the target, while the other one removes the states that can avoid reaching the target. These are qualitative algorithms based on graphs that do not perform any arithmetical operations.

The next section introduces the preliminary concepts we need along the paper. Section 3 presents our method and the proof of correctness. The experiments are shown in Section 4. The last section discusses related results concerning complexity and policy iteration.

2 Preliminaries

We introduce the definitions and known-results used throughout the paper, concerning both Markov decision process and linear programming.

2.1 Markov decision processes

Definition 1. Let $\text{Dist}(A)$ denote the set of discrete probability distributions over the set A . A Markov Decision Process (MDP) M is a pair (S, T) where S is a finite set of *states* and $T \subseteq S \times \text{Dist}(S)$ is a set of *transitions*¹. Given $\mu = (s, d) \in T$, the value $d(t)$ is the probability of making a transition to t from s using μ . We write $\mu(t)$ instead of $d(t)$, and write $\text{state}(\mu)$ for s . We define the set $\text{en}(s)$ as the set of all transitions μ with $\text{state}(\mu) = s$. For simplicity, we make the usual assumption that every state has at least one enabled transition: $\text{en}(s) \neq \emptyset$ for all $s \in S$.

We write $s \xrightarrow{\mu} t$ to denote $\mu \in \text{en}(s) \wedge \mu(t) > 0$. A path in an MDP is a (possibly infinite) sequence $\rho = s^0.\mu^1.s^1.\dots.\mu^n.s^n$, where $\mu^i \in \text{en}(s^{i-1})$ and $\mu^i(s^i) > 0$ for all i . If ρ is finite, the last state of ρ is denoted by $\text{last}(\rho)$, and the length is denoted by $\text{len}(\rho)$ (a path having a single state has length 0). Given a set of states U , we define $\text{reach}(U)$ to be the set of all infinite paths $\rho = s^0.\mu^1.s^1.\dots$ such that $s^i \in U$ for some i .

The semantics of MDPs is given by schedulers. A scheduler η for an MDP M is a function $\eta : S \rightarrow T$ such that $\eta(s) \in \text{en}(s)$ for all s . In words, the scheduler chooses an enabled transition based on the current state. For all schedulers η , $t \in S$, the set $\text{Paths}(t, \eta)$ contains all the paths $s^0.\mu^1.s^1.\dots.\mu^n.s^n$ such that $s^0 = t$, $\mu^i = \eta(s^{i-1})$ and $s^{i-1} \xrightarrow{\mu^i} s^i$ for all i . The reader familiar with MDPs might note that we are restricting to Markovian non-randomized schedulers (that is, they map states to transitions, instead of the more general schedulers mapping paths to distributions on transitions). As explained later on, these schedulers suffice for our purposes.

The probability $\Pr_M^{t,\eta}(\rho)$ of the path ρ under η starting from t is $\prod_{i=1}^{\text{len}(\rho)} \mu^i(s^i)$ if $\rho \in \text{Paths}(t, \eta)$. If $\rho \notin \text{Paths}(t, \eta)$, then the probability is 0. We often omit the subindices M and/or t if they are clear from the context.

We are interested on the probability of (sets of) infinite paths. Given a finite path ρ , the probability of the set ρ^\uparrow comprising all the infinite paths that have ρ as a prefix is defined by $\Pr^\eta(\rho^\uparrow) = \Pr^\eta(\rho)$. In the usual way (that is, by resorting to the Carathéodory extension theorem) it can be shown that the definition on the sets of the form ρ^\uparrow can be extended to σ -algebra generated by the sets ρ^\uparrow .

The verification of PCTL* [4] and ω -regular formulae [6] (for example LTL) can be reduced to the problem of calculating $\max_{s,\eta} \Pr_M^{s,\eta}(\text{reach}(U))$ (or $\min_{s,\eta} \Pr_M^{s,\eta}(\text{reach}(U))$) for MDPs M' , states s and sets U obtained from the formula.

In consequence, in the rest of the paper we concentrate on the following problems.

Definition 2. Given an MDP M , an initial state s and set of *target* states U , a *reachability problem* consists of computing $\max_{s,\eta} \Pr_M^{s,\eta}(\text{reach}(U))$ (or $\min_{s,\eta} \Pr_M^{s,\eta}(\text{reach}(U))$).

From classic results in MDP theory (for these results applied to model checking see, for instance, [1, Chapter 3]) there exists a scheduler η^* such that

$$\eta^* = \arg \max_{\eta} \Pr_M^{s,\eta}(\text{reach}(U)) \quad (1)$$

for all $s \in S$. That is, η^* attains the maximum probability for all states.

An analogous result holds for the the case of minimum probabilities. There exists η^* such that

$$\eta^* = \arg \min_{\eta} \Pr_M^{s,\eta}(\text{reach}(U)) \quad (2)$$

¹Defining transitions as pairs helps to deal with the case in which the same distribution is enabled in several states

for all $s \in S$.

Even in a more general setting allowing for non-Markovian and randomized schedulers, it can be proven that we can assume η^* to be Markovian and non-randomized. The existence of η^* justifies our restriction to Markovian and non-randomized schedulers.

Markov chains A Markov chain (MC) is an MDP such that $|\text{en}(s)| = 1$ for all $s \in S$. Note that a Markov chain has exactly one scheduler, namely the one that chooses the only transition enabled in each state. Hence, for Markov chains, we often disregard the scheduler and denote the probability of reaching U as $\Pr_M^s(\text{reach}(U))$.

Definition 3. Given an MDP $M = (S, T)$ and a scheduler η , we define the Markov chain $M \downarrow \eta = (S, T')$ where $\mu \in T'$ iff $\eta(\text{state}(\mu)) = \mu$.

A simple application of the definitions yields

$$\Pr_M^{s,\eta}(\text{reach}(U)) = \Pr_{M \downarrow \eta}^s(\text{reach}(U)). \quad (3)$$

2.2 Linear programming

We use a particular canonical form of linear programs suitable for our needs. It is based on [5, Appendix B], which is also a good reference for all the concepts and results given in this subsection.

A linear programming problem consists in computing

$$\min_{\mathbf{x}} \{ \mathbf{c}\mathbf{x} \mid A\mathbf{x} = \mathbf{b} \wedge \mathbf{x} \geq \mathbf{0} \}, \quad (4)$$

given a *constraint matrix* A , a *constraint vector* \mathbf{b} and a *cost vector* \mathbf{c} . In the following, we assume that A has m rows and $m+n$ columns, for some $m > 0$ and $n \geq 0$. Hence, \mathbf{c} is a row vector with $m+n$ components, and \mathbf{b} is a column vector with m components.

A *solution* is any vector \mathbf{x} of size $m+n$. The i -th component of \mathbf{x} is denoted by x_i . We say that a solution is *feasible* if $A\mathbf{x} = \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$; it is *optimal* if it is feasible and $\mathbf{c}\mathbf{x}$ is minimum over all feasible \mathbf{x} . A problem is *feasible* if it has a feasible solution, and *bounded* if it has an optimal solution. A non-singular $m \times m$ submatrix of A is called a *basis*. We overload the letter B to denote both the basis and the set of indices of the corresponding columns in A . A variable x_k is *basic* if $k \in B$. Note that, given our assumptions on the dimension of the constraint matrix, for all bases there are m basic variables and n non-basic variables. Given a basis B , and any vector \mathbf{t} , let \mathbf{t}^B be the subvector of \mathbf{t} having only the components in B . When B is clear from the context, we use N to denote the set of columns *not in* B , and use \mathbf{t}^N accordingly. For a matrix A , let A^N be submatrix of A having only the columns that are not in B . The solution \mathbf{x} *induced* by the basis B is defined as $x_k = 0$ for all $k \notin B$, while the values for $k \in B$ are given by the vectorial equation $\mathbf{x}^B = B^{-1}\mathbf{b}$. A solution \mathbf{x} is *basic* if there is a basis that induces \mathbf{x} . Given B and $k \in N$, the *reduced cost* \bar{c}_k of a variable x_k is defined as $c_k - \mathbf{c}^B B^{-1} A_k$, where A_k is the k -th column of A . A solution is *dual feasible* if it correspond to a basis such that $\bar{c}_k \geq 0$ for all $k \in N$.

In our proofs we make use of the following lemma, which is particular to our canonical form.

Lemma 1.

$$A\mathbf{x} = \mathbf{b} \quad \text{if } \mathbf{x} \text{ is basic.}$$

Proof. By splitting A into basic and non-basic columns we get $A\mathbf{x} = B\mathbf{x}^B + A^N\mathbf{x}^N = BB^{-1}\mathbf{b} + A^N\mathbf{0} = I\mathbf{b} = \mathbf{b}$. (Note that \mathbf{x} might not be feasible as it could be $\mathbf{x} \not\geq \mathbf{0}$.) \square

Correctness of the simplex method relies on the following well-known facts about LP problems:

- Every solution that is both feasible and dual feasible is optimal
- If there exists an optimal solution, then there exists a *basic* solution that is feasible and dual feasible (and hence optimal)

As the problems we deal with are ensured to be bounded and feasible, we assume that there exists an optimal solution. In this context, the simplex algorithm explores different bases until it finds a basis whose corresponding solution is feasible and dual feasible.

In several implementations of the algorithm the starting basis can be specified (when it is not, a default one is used). The initial basis does not need to be feasible nor dual feasible. In case the starting basis complies with both feasibilities, the simplex algorithm finishes after checking that these feasibilities are met, without any further exploration. In Subsection 2.3, we show how reachability problems correspond to LP problems. In Section 3 we show that, under a certain assumption on the model checker (Assumption 1), a basis can be obtained from the scheduler provided by the model checker. In particular, optimal schedulers yield feasible bases (Theorem 3). Under our assumption, all the bases obtained from schedulers are dual feasible (Theorem 4).

Among the different variants of the simplex method, in our experiments (Section 4) we use the *dual* simplex, which first looks for a dual-feasible basis (in the so-called *first phase*) and next tries to find a feasible one while keeping dual feasibility (in the *second phase*). This is appropriate in our case since, under our assumptions, the first phase is not needed (as formalized in Theorem 4). In contrast to the dual simplex, the *primal* simplex (or, simply, simplex) looks for a feasible basis in the first phase. As a consequence, if iterations are required (according to our results in Section 3, this is case in which the model checker fails to provide the optimal scheduler), then the primal simplex performs both phases. However, both variants can be used and, as our experiments show, the starting basis obtained from the scheduler is useful to save iterations. In the few cases in which PRISM did not provide the optimal schedulers, the dual simplex required less iterations than the primal one; both of them perform far better when starting from a basis corresponding to a near-optimal scheduler than when starting from the default basis (see Section 4).

2.3 Linear programming for Markov decision processes

Linear programming can be used to compute optimal probabilities for some of the states in the system. The set of states whose maximum (minimum, resp.) probability is 0 is first calculated using graph-based techniques [9, Sec. 4.1]. This qualitative calculation is often considered as a preprocessing step before the proper quantitative model checking. Given a set of target states U , let $S^{\max 0}$ be the set of states S such that $\max_{\eta} \Pr_M^{s,\eta}(\text{reach}(U)) = 0$. Similarly, let $S^{\min 0}$ be the set of states S such that $\min_{\eta} \Pr_M^{s,\eta}(\text{reach}(U)) = 0$. When focusing on maximum probabilities, we write the set $S \setminus (S^{\max 0} \cup U)$ as $S^?$ (called the set of *maybe* states), while for minimum probabilities $S^?$ is $S \setminus (S^{\min 0} \cup U)$.

The maximum probabilities for $s \in S^{\max 0}$ are 0 by definition of $S^{\max 0}$. For $s \in U$ the probabilities are 1, since when starting from a state in U , the set U is reached in the initial state, regardless of the scheduler. The minimum probabilities for $s \in S^{\min 0}$ are 0 by definition of $S^{\min 0}$, and the probabilities for $s \in U$ are again 1. Next we show how to obtain the probabilities for the states in $S^?$, thus covering all the states in the system.

In order to avoid order issues, we assume that the states are $S^? = s_1, \dots, s_n$ and the transitions are:

$$T = \mu_1, \dots, \mu_m \quad (5)$$

in such a way that if $s_i = \text{state}(\mu_j)$, $s_{i'} = \text{state}(\mu_{j'})$ and $i < i'$, then $j < j'$ (from Def. 1, recall that $\text{state}(\mu_i)$ is the state in which μ_i is enabled). Roughly speaking, the transitions are ordered with respect to the states in which they are enabled. From now on, we use this orderings consistently throughout the paper.

In the following theorem, the matrix $A|I$ associated to a reachability problem $\max \Pr_M^{s_i, \eta}(\text{reach}(U))$ is a $m \times (n + m)$ matrix whose last m columns form the identity matrix. We define of $A_{i,j}$ for the column $j \leq n$ as: $A_{i,j} = \mu_i(s_j)$ if $s_j \neq \text{state}(\mu_i)$, or $A_{i,j} = \mu_i(s_j) - 1$ if $s_j = \text{state}(\mu_i)$. The vector \mathbf{b} is defined as $b_i = -\sum_{s \in U} \mu_i(s)$.

Theorem 1. *For all states $s_i \in S^?$, the value $\max_{\eta} \Pr_M^{s_i, \eta}(\text{reach}(U))$ is the value of the variable x_i in an optimal solution of the following LP problem:*

$$\begin{aligned} \min \quad & \overbrace{(1, \dots, 1, 0, \dots, 0)}^n \mathbf{x} \\ & (A \mid I^{m \times m}) \mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}. \end{aligned} \quad (6)$$

Analogously, the value $\min_{\eta} \Pr_M^{s_i, \eta}(\text{reach}(U))$ is the value of the variable x_i in an optimal solution of the following LP problem.

$$\begin{aligned} \min \quad & -\overbrace{(1, \dots, 1, 0, \dots, 0)}^n \mathbf{x} \\ & (-A \mid I^{m \times m}) \mathbf{x} = -\mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}. \end{aligned} \quad (7)$$

(Note that, in the constraint, the matrix A is negated, while I is not.)

This theorem is just the well-known correspondence between reachability problems and LP problems [11],[9, Section 4.2], written in our LP setting.

The variables that multiply the columns in the identity matrix are called *slack* variables in the LP literature. They are also the variables x_{μ} in the following notation.

Notation 1. *From now on, we identify each column $1 \leq j \leq n$ of $(A|I)$ with the state s_j , and each column $n < j \leq n + m$ with the transition μ_j . Each row i is identified with μ_i . In consequence, we write $A_{\mu, s}$ for the elements of the matrix, and x_s or x_{μ} for the components of the solution \mathbf{x} .*

3 A method for exact solutions

Our method serves as a complement to a model checker being able to:

- calculate the set $S^?$, and
- give a description of a scheduler, that the model checker *considers* optimal based on finite precision calculations

We only require a weak “optimality” condition on the scheduler returned by the model checker, which we refer to as *apt*: we say that a scheduler η is apt iff $\Pr_M^{s, \eta}(\text{reach}(U)) > 0$ for all $s \in S^?$. In other words, we only require the scheduler to reach U for all states that can reach it (no matter with which probability). In the case of minimum probabilities, every scheduler is apt, since if we have $\Pr_M^{s, \eta}(\text{reach}(U)) = 0$ for some

η , then $s \notin S^?$ (by definition of $S^{\min 0}$). For the case of the maximum, the existence of an apt scheduler follows from the definition of $S^?$, the scheduler η^* in (1) being a suitable witness.

Assumption 1. *We assume that the model checker is able to provide an apt scheduler, in the sense that our method is not guaranteed to return a value in case the scheduler is not apt.*

Our method is described in the Algorithm 1. The function `construct_problem` constructs the LP

<p>input : An MDP M and a set of states U</p> <p>output: \mathbf{x} such that $x_s = \max_{\eta} \Pr_M^{s,\eta}(\text{reach}(U))$ ($\min_{\eta} \Pr_M^{s,\eta}(\text{reach}(U))$, resp.) for all $s \in S^?$</p> <pre> 1 // Use model checker to get the set $S^?$ and a scheduler 2 $(S^?, \eta) \leftarrow \text{reach_analysis}(M, U)$; 3 $\mathcal{L} \leftarrow \text{construct_problem}(M, S^?)$; 4 $B_{\eta} \leftarrow \text{construct_basis}(\mathcal{L}, \eta)$; 5 $\text{start_simplex_solver}(\mathcal{L}, B_{\eta})$; 6 if the exact simplex solver finishes in one iteration then 7 return $\arg \min_{\mathbf{x}} \mathcal{L}$, obtained from the solver; 8 else if the solver performs several iterations then // η is not optimal 9 return $\arg \min_{\mathbf{x}} \mathcal{L}$, obtained from the solver once it finishes; 10 // Or interrupt the solver and change the model checker parameters 11 else if the solver reports that the basis is singular then 12 // For the minimum, this case cannot happen 13 error η is not apt; 14 end </pre>
--

Algorithm 1: Method to get exact solutions

problems (6) and (7). Given η , the basis B_{η} obtained by `construct_basis` defined as

$$s \in B_{\eta}, \quad \text{for all } s \in S^? \quad x_{\mu} \in B_{\eta} \iff \eta(\text{state}(\mu)) \neq \mu. \quad (8)$$

Roughly speaking, the basis contains all states, and all the transitions that are *not* chosen by η . Sometimes (particularly in the proof of Theorem 4) we write $B_{M',\eta}$ to make it clear that the basis belongs to an MDP M' .

The rest of this section is devoted to prove the correctness of the algorithm, in the sense made precise by the following theorem (which is proven later).

Theorem 2. *If the algorithm returns a value, then the value corresponds to the **output** specification. Moreover, if the scheduler η provided by the model checker is apt, then the matrix defined by (8) is a basis, and the algorithm returns optimum values from the LP solver. If the scheduler provided by the model checker is optimum as in (1), then the basis in (8) is both feasible and dual feasible.*

Recall from Subsection 2.2 that the simplex algorithm stops as soon as it finds a solution that is feasible and dual feasible. Hence, the fact that an optimal scheduler yields a basic, feasible and dual feasible solution causes the simplex solver to stop as soon as the feasibility checks are finished.

The rest of this section is devoted to prove Theorem 2. In our proofs we resort to the following definitions and lemmata. The first definition uses indices as explained in Notation 1.

Definition 4. Given a scheduler η , we write the set of transitions complying with $\eta(\text{state}(\mu)) = \mu$ as $T_\eta = \{\mu^1, \dots, \mu^n\}$, and we assume that this ordering respects the ordering in (5). We define C^η to be the $n \times n$ matrix whose elements are as $C_{i,j}^\eta = \mu^i(s_j)$. Consider the matrix A in (6). We define $(A \downarrow \eta)$ to be the $n \times n$ submatrix of A comprising all the rows $\mu \in T_\eta$ and the columns s for all $s \in S^?$.

Lemma 2. *The transitions $\mu^i \in T_\eta$ comply with $\text{state}(\mu^i) = s_i$ for all $s_i \in S^?$. In consequence, $\eta(s_i) = \mu^i$.*

Proof. Note that since the order in T_η respects the order in (5), we have that the sequence $\text{state}(\mu^1), \dots, \text{state}(\mu^n)$ is a sequence of states s_{j_1}, \dots, s_{j_n} with $j_1 \leq \dots \leq j_n$. Since there are n states, and for each state s we have exactly one transition μ such that $\eta(s) = \mu$, it must be $s_{j_1} = s_1, \dots, s_{j_n} = s_n$. This implies $\text{state}(\mu^i) = s_{j_i} = s_i$ as desired. Using this equality and $\mu^i \in T_\eta$ we have $\eta(s_i) = \eta(\text{state}(\mu^i)) = \mu^i$. \square

Lemma 3. *For all η , we have $(A \downarrow \eta) = C^\eta - I$.*

Proof. By definition of $(A \downarrow \eta)$ and the definition of the matrix A in (6) we have $(A \downarrow \eta)_{i,j} = A_{\mu^i, s_j} = \mu^i(s_j) - Q_{i,j}$, where $Q_{i,j} = 1$ if $\text{state}(\mu^i) = s_j$, or otherwise $Q_{i,j} = 0$. By Lemma 2, we have $\text{state}(\mu^i) = s_j$ iff $i = j$. Hence $Q_{i,j}$ is the identity matrix and $(A \downarrow \eta)_{i,j} = \mu^i(s_j) - I_{i,j} = C_{i,j}^\eta - I_{i,j}$, which completes the proof. \square

The matrix $(A \downarrow \eta)$ happens to be very important in our proofs. We profit from the fact that it is non-singular provided that η is apt.

Lemma 4. *For all apt η , the matrix $(A \downarrow \eta)$ is non-singular.*

Proof. Suppose, towards a contradiction, that there exists $\mathbf{x} \neq \mathbf{0}$ such that $(A \downarrow \eta)\mathbf{x} = \mathbf{0}$. Then, by Lemma 3, we have $(C^\eta - I)\mathbf{x} = \mathbf{0}$, which implies $C^\eta \mathbf{x} = \mathbf{x}$ and hence $(C^\eta)^z \mathbf{x} = \mathbf{x}$ for all $z \geq 0$. We arrive to a contradiction by showing that for all j there exists z such that

$$|((C^\eta)^z \mathbf{x})_j| < \max_{s'} |x_{s'}|. \quad (9)$$

In particular, for $q = \arg \max_{s'} |x_{s'}|$ this yields $|((C^\eta)^z \mathbf{x})_q| < |x_q|$, which contradicts $(C^\eta)^z \mathbf{x} = \mathbf{x}$.

Now we prove (9). Since η is apt, from every $s_j \in S^?$ there exists a path $\rho \in \text{Paths}(s_j, \eta)$ with $\text{last}(\rho) \in U$, such that all the states previous to $\text{last}(\rho)$ are not in U . We prove that z can be taken to be $\text{len}(\rho)$. We proceed by induction on the length of ρ . If $\text{len}(\rho) = 1$, by Lemma 2 we have $\eta(s_j)(u) = \mu^j(u) > 0$ for some $u \in U$, and hence² $\sum_{t \in S^?} \mu^j(t) < 1$. Taking $z = 1$ we obtain

$$|(C^\eta \mathbf{x})_j| = \left| \sum_{t \in S^?} \mu^j(t) x_t \right| \leq \sum_{t \in S^?} \mu^j(t) |x_t| \leq \sum_{t \in S^?} \mu^j(t) \max_{s'} |x_{s'}| < \max_{s'} |x_{s'}|,$$

which proves that we can take $z = 1 = \text{len}(\rho)$. The last strict inequality holds only if $\max_{s'} |x_{s'}| > 0$, which follows from $\mathbf{x} \neq \mathbf{0}$.

²The result for discounted MDPs does not use $S^?$ as the analogous of this sum is always less than 1 due to the discounts

If $\text{len}(\rho) = l + 1$, there exists $s_q \in S^?$ such that $\mu^j(s_q) > 0$ and q reaches U in l steps. The inductive hypothesis holds for q , and hence $|((C^\eta)^l \mathbf{x})_q| < \max_{s'} |x_{s'}|$, from which we obtain:

$$\begin{aligned} |((C^\eta)^{l+1} \mathbf{x})_j| &= |(C^\eta (C^\eta)^l \mathbf{x})_j| \leq \sum_{t \in S^? \setminus \{s_q\}} \mu^j(t) |((C^\eta)^l \mathbf{x})_t| + \mu^j(s_q) |((C^\eta)^l \mathbf{x})_q| \\ &= \sum_{t \in S^? \setminus \{s_q\}} \mu^j(t) |x_t| + \mu^j(s_q) |((C^\eta)^l \mathbf{x})_q| \leq \sum_{t \in S^? \setminus \{s_q\}} \mu^j(t) \max_{s'} |x_{s'}| + \mu^j(s_q) |((C^\eta)^l \mathbf{x})_q| \\ &< \sum_{t \in S^? \setminus \{s_q\}} \mu^j(t) \max_{s'} |x_{s'}| + \mu^j(s_q) \max_{s'} |x_{s'}| \leq \max_{s'} |x_{s'}| \end{aligned}$$

This finishes the proof of (9). Assuming that $(A \downarrow \eta) \mathbf{x} = \mathbf{0}$ for some $\mathbf{x} \neq \mathbf{0}$, we derived (9), which contradicts $(C^\eta)^z \mathbf{x} = \mathbf{x}$ for all $z \geq 0$, thus finishing the proof. \square

Lemma 5. *For all apt schedulers η , the basis defined in (8) is non-singular.*

Proof. We show that the equation $B_\eta \mathbf{x} = \mathbf{0}$ holds only if $\mathbf{x} = \mathbf{0}$. Note that the vector \mathbf{x} has one component for each column of the basis, that is, one component for each state in $S^?$ (called x_s), and one component for each transition such that $\eta(\text{state}(\mu)) \neq \mu$ (called x_μ). The matrix equation $B_\eta \mathbf{x} = \mathbf{0}$ corresponds to m equations, one for each transition. If $\mu \in B_\eta$, since $t \in B_\eta$ for all $t \in S^?$, the equation corresponding to μ is

$$\sum_{t \in S^?} A_{\mu,t} x_t + x_\mu = 0. \quad (10)$$

If $\mu \notin B_\eta$, the corresponding equation is

$$\sum_{t \in S^?} A_{\mu,t} x_t = 0. \quad (11)$$

(Note that the sum term x_μ has disappeared. This corresponds to the fact that the column corresponding to x_μ is not in the basis.) Since the transitions $\mu \notin B_\eta$ are those such that $\eta(\text{state}(\mu)) = \mu$, the set of equations (11) is equivalent to $(A \downarrow \eta) \mathbf{s} = \mathbf{0}$, where \mathbf{s} is the subvector of \mathbf{x} having only the components corresponding to states. In consequence, if $B_\eta \mathbf{x} = \mathbf{0}$ holds, then in particular $(A \downarrow \eta) \mathbf{s} = \mathbf{0}$ and, since η is apt, by Lemma 4 it must be $\mathbf{s} = \mathbf{0}$, that is, $x_t = 0$ for all $t \in S^?$. Using this in (10) we have $x_\mu = 0$ for all $\mu \in B_\eta$. We have proven $x_j = 0$ for every component j of \mathbf{x} , thus showing $\mathbf{x} = \mathbf{0}$. \square

Theorem 3. *If a scheduler is optimal as in (1) (or (2), resp.) then the solution induced by the basis B_η is feasible.*

Proof. Let \mathbf{x} be the solution induced by B_η for some optimal η . By Lemma 1, we need to prove $\mathbf{x} \geq \mathbf{0}$. We prove this inequality by showing that $x_s = \text{Pr}_M^{s,\eta}(\text{reach}(U)) \geq 0$ for all s and $x_\mu \geq 0$ for all μ .

Since in B_η the variables $x_\mu \in T_\eta$ are non basic, in the solution \mathbf{x}^η induced by B_η we have $x_\mu = 0$ for all $\mu \in T_\eta$. Then, using Lemma 1 for our particular constraint matrix $A|I$, we obtain

$$x_s = \sum_{t \in S^?} \eta(s)(t) x_t + \sum_{t \in U} \eta(s)(t). \quad (12)$$

This is equivalent to $(A \downarrow \eta) \mathbf{x} = \mathbf{q}$ for some vector \mathbf{q} . By Lemma 4, there exists exactly one \mathbf{x} satisfying (12). Let v_s^η be $\text{Pr}_M^{s,\eta}(\text{reach}(U))$. A classic result for MDPs (see, for instance, [9, Section 4.2], [1,

Theorem 3.10]) states that, for an optimal scheduler η , it holds

$$v_s^\eta = \max_{\mu \in \text{en}(s)} \sum_{t \in S^?} \mu(t) v_t^\eta + \sum_{t \in U} \mu(t) \quad (13)$$

and

$$\eta(s) \in \arg \max_{\mu \in \text{en}(s)} \sum_{t \in S^?} \mu(t) v_t^\eta + \sum_{t \in U} \mu(t) .$$

for all states s . From the last two equations:

$$v_s^\eta = \sum_{t \in S^?} \eta(s)(t) v_t^\eta + \sum_{t \in U} \eta(s)(t) .$$

This is equivalent to $(A \downarrow \eta) \mathbf{v}^\eta = \mathbf{q}$ as before. After (12) we have seen that this equation has a unique solution, and so $x_s = v_s^\eta$ for all $s \in S^?$. By (13) we have

$$x_s \geq \sum_{t \in S^?} \mu(t) x_t + \sum_{t \in U} \mu(t) \quad (14)$$

for all $s \in S^?$, $\mu \in \text{en}(s)$. Applying Lemma 1 to our particular constraint matrix $A|I$, we have

$$x_\mu = x_s - \sum_{t \in S^?} \mu(t) x_t - \sum_{t \in U} \mu(t) .$$

Hence, $x_\mu \geq 0$ for all μ by (14). In conclusion, $x_s = v_s^\eta \geq 0$ for all $s \in S^?$ and $x_\mu \geq 0$ for all μ . Then, the solution \mathbf{x} induced by B_η is feasible.

For the case of the minimum, the analogue of (13) is:

$$v_s^\eta = \min_{\mu \in \text{en}(s)} \sum_{t \in S^?} \mu(t) v_t^\eta + \sum_{t \in U} \mu(t) \quad (15)$$

The fact that the equation $(A \downarrow \eta) \mathbf{v}^\eta = \mathbf{q}$ has a unique solution again yields $x_s = \mathbf{v}^\eta$. For x_μ , using the constraint matrix $-A|I$ for the minimum and (15) we obtain

$$x_\mu = -x_s + \sum_{t \in S^?} \mu(t) x_t + \sum_{t \in U} \mu(t) = \sum_{t \in U} \mu(t) + \sum_{t \in S^?} \mu(t) - x_t \geq 0 .$$

□

Theorem 4. *Given an apt scheduler η , the solution induced by the basis B_η is dual feasible. (For the definition of dual feasible see Subsection 2.2.)*

Proof. First we find a matrix expression for B_η^{-1} . Suppose we reorder the rows of B_η so that the rows corresponding to transitions in the basis occur first. The resulting matrix is

$$B'_\eta = \left(\begin{array}{c|c} A' & I^{(m-n) \times (m-n)} \\ \hline (A \downarrow \eta) & 0 \end{array} \right)$$

where A' is a submatrix of B_η . We can write

$$B'_\eta = P B_\eta \quad (16)$$

where P is a permutation matrix. In order to find the inverse of B'_η we pose the following matrix equation:

$$\left(\begin{array}{c|c} A' & I^{(m-n) \times (m-n)} \\ \hline (A \downarrow \eta) & 0 \end{array} \right) \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|c} A'A_{11} + A_{21} & A'A_{12} + A_{22} \\ \hline (A \downarrow \eta)A_{11} & (A \downarrow \eta)A_{12} \end{array} \right) = I = \left(\begin{array}{c|c} I^{n \times n} & 0 \\ \hline 0 & I^{(m-n) \times (m-n)} \end{array} \right)$$

These equations, suggest that we can take $A_{11} = 0$, and hence $A_{21} = I$. Moreover, it must be $A_{12} = (A \downarrow \eta)^{-1}$ (which exists by Lemma 5) and hence $A_{22} = -A'(A \downarrow \eta)^{-1}$. The equation below can be easily checked by verifying that $B'^{-1}_\eta B'_\eta = I$

$$B'^{-1}_\eta = \left(\begin{array}{c|c} 0^{n \times (m-n)} & (A \downarrow \eta)^{-1} \\ \hline I^{(m-n) \times (m-n)} & -A'(A \downarrow \eta)^{-1} \end{array} \right) \quad (17)$$

Next, we use (17) to show that the reduced costs depend only on the constraint coefficients of the transitions chosen by the scheduler.

We consider first the case of the maximum. Recall that our constraint matrix is $A|I$ and the costs c_μ associated to the transitions variables are 0 for all μ (see (6)). According to the definition of reduced cost (see Subsection 2.2), to prove dual feasibility we need to show $-\mathbf{c}^{B_\eta} B_\eta^{-1} I_\mu \geq 0$ for all $\mu \notin B_\eta$, where I_μ is the column of the identity matrix corresponding to μ . From (16), we have $B_\eta^{-1} = B'^{-1}_\eta P$, and hence our inequality is $-\mathbf{c}^{B_\eta} B'^{-1}_\eta P I_\mu \geq 0$. Since P is a permutation matrix, we know that $P I_\mu$ is a column of the identity matrix, say $I_{k(\mu)}$. Given our costs in (6), and given the definition of B_η , we have that \mathbf{c}^{B_η} is the vector $(\overbrace{1, \dots, 1}^n, \overbrace{0, \dots, 0}^{m-n})$, and hence from (17) we get $\mathbf{c}^{B_\eta} B'^{-1}_\eta = (\mathbf{0}^{1 \times (m-n)}, \mathbf{1}^{1 \times n} (A \downarrow \eta)^{-1})$. In conclusion, we have proven

$$-\mathbf{c}^{B_\eta} B_\eta^{-1} I_\mu = -(\mathbf{0}^{1 \times (m-n)}, \mathbf{1}^{1 \times n} (A \downarrow \eta)^{-1}) I_{k(\mu)}, \quad (18)$$

and we must prove that this number is greater than or equal to 0 for all $\mu \notin B_\eta$.

Whenever $k(\mu) \leq m - n$, the result holds since (18) is 0.

In case $k(\mu) > m - n$, we prove the result using the fact that these values depend only on the transitions chosen by η . In fact, given the MDP M and the scheduler η , if we write (18) for the Markov chain $M \downarrow \eta$ (see Def. 3), we obtain

$$-\mathbf{c}^{B_{(M \downarrow \eta), \eta}} B_{(M \downarrow \eta), \eta}^{-1} I_\mu = -\mathbf{1}^{1 \times n} (A \downarrow \eta)^{-1} I_\mu \quad (19)$$

for all $\mu \notin B_{(M \downarrow \eta), \eta}$. Note that for $M \downarrow \eta$ there is no need to reorder (as there are no transitions in the basis) and so $\mu = k(\mu)$. Given that all the transitions $M \downarrow \eta$ are chosen by η , the basis $B_{(M \downarrow \eta), \eta}$ contains all the states and no transitions. In this equation, I_μ can be *any* column of $I^{n \times n}$ (again, due to the fact that there are no transitions in the basis).

Suppose, towards a contradiction, that (18) is less than 0 for some $k(\mu) > m - n$. This is equivalent to $-\mathbf{1}^{1 \times n} (A \downarrow \eta)^{-1} I_{k(\mu) - (m-n)} < 0$. By (19) we have $-\mathbf{1}^{1 \times n} (A \downarrow \eta)^{-1} I_{\mu'} < 0$ for some μ' in $M \downarrow \eta$. Then, the solution induced by the basis is not dual feasible for the problem associated to $M \downarrow \eta$. As there is at least one optimal basic and dual feasible solution (the one found by simplex method), there exists an optimal solution \mathbf{x}^C such that the corresponding basis B^C is not $B_{(M \downarrow \eta), \eta}$. As in $M \downarrow \eta$ there

exists only one basis containing all states (namely $B_{(M \downarrow \eta), \eta}$), there exists $s \notin B^C$. In consequence, we have $x_s^C = 0$. Since x^C is optimal, by Theorem 1, we obtain $\Pr_{M \downarrow \eta}^{s, \eta}(\text{reach}(U)) = 0$, from which (3) yields $\Pr_M^{s, \eta}(\text{reach}(U)) = 0$. This contradicts the fact that η is apt.

The proof for the case of the minimum is completely analogous: despite the differences in the constraints and the cost vector, the reduced costs in (18) are the same as before:

$$-\mathbf{c}^{B_\eta} B_\eta^{-1} I_\mu = -(-\mathbf{0}^{1 \times (m-n)}, -\mathbf{1}^{1 \times n} (-(A \downarrow \eta)^{-1})) I_{k(\mu)} = -(\mathbf{0}^{1 \times (m-n)}, \mathbf{1}^{1 \times n} (A \downarrow \eta)^{-1}) I_{k(\mu)}.$$

These values again coincide with the ones in a system having only the transitions chosen by η . □

Proof (of Theorem 2). If the algorithm returns a value, then it is $\arg \min_{\mathbf{x}} \mathcal{L}$, where \mathcal{L} is the problem (6) (or (7) for the minimum). Hence, by Theorem 1, the returned value coincides with the **output** specification. We have that if η is apt, then B_η is a basis by Lemma 5. As a consequence, the algorithm never enters the branch in line 11, and so the result is returned. The termination in a single iteration is a consequence of the fact that the solution corresponding to an optimal scheduler η is both feasible (Theorem 3) and dual feasible (Theorem 4). □

4 Experimental results

Implementation. We implemented our method by extending the model checker PRISM [10], using the LP library glpk [2]. We compiled glpk using the library for arbitrary precision gmp. We needed to modify the code of glpk: although there is a solver function that uses exact arithmetic internally, this function does not allow us to retrieve the exact value. Aside from these changes to glpk and some additional code scattered around the PRISM code (in order to gather information about the scheduler), the specific code for implementing our method is less than 300 lines long. With these modifications, PRISM is able to print the numerator and the denominator of the probabilities calculated.

Our implementation works as follows: in the first step, we use the value iteration already implemented in PRISM to calculate a candidate scheduler. In the next step, the LP problem is constructed by iterating over each state: for each transition enabled, the corresponding probabilities are inserted in the matrix. The basis is constructed along this process: when a transition is considered, the description of the scheduler (implemented as an array) is queried about whether this transition is the one chosen by the scheduler. Next we solve the LP problem. For the reasons explained in Subsection 2.2, in Algorithm 1 we use the dual simplex method, except when we compare it to the primal one. The reader familiar with glpk might notice that the dual variant is not implemented under exact arithmetic on glpk: to overcome this, instead of providing glpk with the original problem, we provided the dual problem and retrieved the values of the dual variables (the dual problem is obtained by providing the transpose of the constraint matrix and by negating the cost coefficients, and so it does not affect the running time).

The experiments were carried out on an Intel i7 @3.40Ghz with 8Gb RAM, running Windows 7.

Case studies. We studied three known models available from the PRISM benchmark suite [14], where the reader can look for matters not explained here (for instance, details about the parameters of each model). For the parameters whose values are not specified here, we use the default values. In the IEEE 802.11 Wireless LAN model, two stations use a randomised exponential backoff rule to minimise the likelihood of transmission collision. The parameter N is the number of maximum backoffs. We compute the maximum probability that the backoff counters of both stations reach their maximum value. The

Model	Para- meters	$n = S^2 $	m	Time (seconds)					
				LP without Alg. 1		Algorithm 1			
				Primal	Dual	Value iter.	LP constr.	Dual simplex	Total
Wlan (N)	3	2529	96302	19.53	11.76	0.36	0.05	0.03	0.44
	4	5781	345000	110.32	61.83	2.30	0.21	0.06	2.57
	5	12309	1295218	535.76	326.64	14.93	1.32	0.15	16.40
Consensus (N, K)	3,3	3607	3968	251.74	35.32	2.93	0.04	0.15	3.12
	3,4	4783	5216	488.84	64.00	6.47	0.06	0.58	7.11
	3,5	5959	6464	1085.70	105.36	12.74	0.06	1.87	14.67
	4,1	11450	12416	-	432.98	2.88	0.11	0.19	3.18
	4,2	21690	22656	-	1951.91	20.41	0.23	0.37	21.01
	4,3	31930	32896	-	-	59.73	0.49	0.58	60.80
	4,4	42170	43136	-	-	134.62	0.64	0.78	136.04
	4,5	52410	53376	-	-	246.90	0.91	0.96	248.77
Firewire (D)	200	1071	80980	4.50	2.65	0.28	0.04	0.01	0.33
	300	23782	213805	-	1314.32	2.89	1.04	0.24	4.17
	400	81943	434364	-	-	11.05	8.74	0.88	20.67

Table 1: Comparison of primal and dual simplex starting from a default basis against Algorithm 1

second model concerns the consensus algorithm for N processes of Aspnes & Herlihy [3], which uses shared coins. We calculate the maximum probability that the protocol finishes without an agreement. The parameter K is used to bound a shared counter. Our third case study is the IEEE 1394 FireWire Root Contention Protocol (using the PRISM model which is based on [12]). We calculate the minimum probability that a leader is elected before a deadline of D time units.

Linear programming versus Algorithm 1. Table 1 allows us to compare (primal and dual) simplex starting from a default basis, against Algorithm 1, which provides a starting basis from a candidate scheduler. Aside from the construction of the MDP from the PRISM language description (which is the same either using LP or Algorithm 1, and is thus disregarded in our comparisons), the steps in our implementation are: (1) perform value iteration to obtain a candidate scheduler; (2) construct the LP problem; (3) solve the problem in exact arithmetic in zero or more iterations (the latter is the case in which the scheduler is not optimal). All these times are shown in Table 1, as well as its sum, expressed in seconds. The experiments for LP were run with a time-out of one hour (represented with a dash).

Our method always outperforms the naive application of LP. The case with the lowest advantage is Consensus (3,5), and still our method takes less than 1/6 of the time required by dual simplex.

With respect to the time devoted to exact arithmetic in Algorithm 1, in all cases the simplex under exact arithmetic takes a fraction of the time spent by the other operations of the algorithm (namely, to perform value iteration and to construct the LP problem). In Consensus (3,5), the simplex algorithm takes less than 1/6 of the time devoted to the other operations. In all other cases the ratio is even lower.

The greatest number found was 28821938103543398400, the denominator in the solution of Firewire 400. It needs 65 bits to be stored. The computations were performed using 32 bit libraries, and so the exact arithmetic computations used around 3 words in the worst case (which is not really a challenge for an arbitrary precision library). We can conclude that, even for systems with more than 10000 states (up to 80000, in our experiments), the overhead introduced by exact arithmetic is manageable.

Suboptimal schedulers as suboptimal bases. Other than measuring whether the calculation is reasonably quick in case the scheduler from PRISM is optimal, a secondary measuring concerns how close is the basis to an optimal one in case the scheduler provided by PRISM is *not* optimal.

	Primal						Dual					
	Iterations			Time (seconds)			Iterations			Time (seconds)		
$\varepsilon (10^{-n})$	6	7	16	6	7	16	6	7	16	6	7	16
Consensus (3,3)	187	134	0	3.43	2.43	0.10	10	6	0	0.19	0.15	0.10
Consensus (3,4)	2497	6278	0	74.42	202.58	0.14	37	28	0	0.63	0.51	0.13
Consensus (3,5)	4990	4340	1239	190.53	160.44	49.487	94	61	6	1.93	1.24	0.25

Table 2: Time spent when the starting basis is not optimal

Except in cases Consensus (3, \cdot), simplex stopped after 0 iterations, thus indicating that PRISM was able to find the optimal scheduler. For optimal schedulers there is no difference between using primal or dual simplex in Algorithm 1 (we ran the experiments and the running time of the simplex variants differed by at most 0.05 seconds).

The probabilities obtained in each step of the value iteration converge to those of an optimal scheduler. Given a threshold ε , value iteration stops only after $|x_s - x'_s| \leq \varepsilon$ for all s , where \mathbf{x} and \mathbf{x}' are the vectors obtained in the last two iterations.

In Table 2 we compare the amount of iterations and the time spent by primal and dual simplex for schedulers obtained using different thresholds. We considered only the cases Consensus (3, \cdot), as in other cases the scheduler returned by PRISM was optimum except for gross thresholds above 0.05, which are rarely used in practice (the default ε in PRISM is 10^{-6}). In addition to the default value, we considered representatives the value 10^{-7} (since 10^{-8} already yields the exact solution for (3,3) in the dual case: a value smaller than 10^{-7} would have yielded uninteresting numbers for this case) and the value 10^{-16} , since in (3,5) the scheduler does not improve beyond such threshold. In fact, for 10^{-16} the result is the same as for 10^{-323} , and 10^{-324} is not a valid double. In Java, the type double corresponds to a IEEE 754 64-bit floating point.

In consequence, we have one case (namely, Consensus (3,5)), where PRISM cannot find the worst-case scheduler for any double threshold (and thus should be recoded to use another arithmetic primitives to get exact results), while our method is able to calculate exact results using less than two seconds after value iteration, as shown in Table 1.

For Consensus (3, \cdot) we see that dual simplex performs better than primal simplex. Consensus (3,4) shows that the primal simplex can behave worse when starting from B_η than the dual simplex starting from the default basis (compare with the corresponding row in Table 1). Moreover, it can be the case that it takes *more* time as the threshold decreases (note that, in contrast, in Consensus (3,5) the time decreases with the threshold, as expected). This suggests that the dual variant should be preferred over the primal.

Comparing against Table 1, we see that, for each variant of the simplex method, starting from the basis B_η results in a quicker calculation than starting from the default basis.

It is worth mentioning that in all cases the difference between the probabilities provided by PRISM and the exact values was less than the threshold for value iteration. It indicates that the finite precision of the computations does not affect the results significantly.

5 Discussion and further work

Linear programming versus policy iteration. It is known that the dual simplex method applied for discounted MDPs is just the same as *policy iteration* (for an introduction to this method see [9]) seen from

a different perspective. Indeed, this has been used to obtain complexity bounds (see [13]). Theorems 3 and 4 establish for undiscounted MDPs the same correspondence between basis and schedulers as known for the discounted case, and as a consequence the dual simplex is policy iteration disguised, also in the undiscounted case.

Even without considering the results in this paper at all, exact solutions can also be calculated by implementing policy iteration with exact arithmetic as, in each iteration, the method calculates the probabilities corresponding to a scheduler and checks whether they can be improved by another scheduler. Roughly speaking, if the calculation and the check are performed using exact arithmetic, then the result is also exact.

Despite this existing alternative, the correspondence between bases and schedulers we presented in this paper allows to obtain an exact solution by using LP solvers, thus profiting from all the knowledge concerning LP problems (and from existing implementations such as `glpk`).

Complexity. To the best of our knowledge, the precise complexity of the simplex method in our case is unknown. There are recent results for the simplex applied to similar problems. For instance, in [13] it is proven that simplex is strongly polynomial for discounted MDPs. Nevertheless, [8] shows an exponential lower bound to calculate rewards in the undiscounted case. Unfortunately (or not, as there is still hope that we can prove the time to be polynomial in our case), the construction used in [8] cannot be carried out easily to our setting, as some of the rewards in the construction are negative (and the equivalent to the rewards in our setting are the sums $\sum_{t \in U} \mu(t)$).

Further work. In the comparison of our method against LP, we considered only the simplex method, as `glpk` only implements this method in exact arithmetic. The feasibility/applicability of other algorithms to solve LP problems using exact arithmetic is yet to be studied.

The fact that the probabilities obtained are exact allows to prove additional facts about the system under consideration. For instance, the exact values can be used in correctness certificates, or be the input of automatic theorem provers, if they require exact values to prove some other properties of the system. We plan to concentrate on these uses of exact probabilities.

Acknowledgements. The author is grateful to David Parker, Vojtech Forejt and Marta Kwiatkowska for useful comments and proofreading.

References

- [1] Luca de Alfaro (1998): *Formal Verification of Probabilistic Systems*. Thesis CS-TR-98-1601, Stanford University, Department of Computer Science.
- [2] Copyright by Andrew Makhorin: *glpk*. <http://www.gnu.org/s/glpk/>.
- [3] James Aspnes & Maurice Herlihy (1990): *Fast Randomized Consensus Using Shared Memory*. *J. Algorithms* 11(3), pp. 441–461. Available at [http://dx.doi.org/10.1016/0196-6774\(90\)90021-6](http://dx.doi.org/10.1016/0196-6774(90)90021-6).
- [4] Andrea Bianco & Luca de Alfaro (1995): *Model Checking of Probabilistic and Nondeterministic Systems*. In: *FSTTCS 1995*, pp. 499–513. Available at http://dx.doi.org/10.1007/3-540-60692-0_70.
- [5] Stephen P. Bradley, Arnoldo C. Hax & Thomas L. Magnanti (1977): *Applied Mathematical Programming*. Addison-Wesley.

- [6] Costas Courcoubetis & Mihalis Yannakakis (1990): *Markov Decision Processes and Regular Events (Extended Abstract)*. In Mike Paterson, editor: *ICALP, Lecture Notes in Computer Science* 443, Springer, pp. 336–349. Available at <http://dx.doi.org/10.1007/BFb0032043>.
- [7] F. D'Epenoux (1963): *A probabilistic production and inventory problem*. *Management Science* 10, pp. 98–108.
- [8] John Fearnley (2010): *Exponential Lower Bounds for Policy Iteration*. In: *ICALP 2010 (2)*, pp. 551–562. Available at http://dx.doi.org/10.1007/978-3-642-14162-1_46.
- [9] Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman & David Parker (2011): *Automated Verification Techniques for Probabilistic Systems*. In: *SFM*, pp. 53–113. Available at <http://dx.doi.org/10.1007/978-3-642-21455-4>.
- [10] M. Kwiatkowska, G. Norman & D. Parker (2011): *PRISM 4.0: Verification of Probabilistic Real-time Systems*. In G. Gopalakrishnan & S. Qadeer, editors: *CAV 2011, LNCS* 6806, Springer, pp. 585–591. Available at http://dx.doi.org/10.1007/978-3-642-22110-1_47.
- [11] Martin L. Puterman (1994): *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st edition. John Wiley & Sons, Inc., New York, NY, USA.
- [12] M. Stoelinga & F. Vaandrager (1999): *Root Contention in IEEE 1394*. In J.-P. Katoen, editor: *Proc. 5th International AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)*, *LNCS* 1601, Springer, pp. 53–74. Available at http://dx.doi.org/10.1007/3-540-48778-6_4.
- [13] Yinyu Ye (2011): *The Simplex and Policy-Iteration Methods Are Strongly Polynomial for the Markov Decision Problem with a Fixed Discount Rate*. *Mathematics of Operations Research* 36(4), pp. 593–603. Available at <http://dx.doi.org/10.1287/moor.1110.0516>.
- [14] <http://www.prismmodelchecker.org/benchmarks/>.